

NAT'L INST. OF STAND & TECH R.I.C.



A11104 232756

NATIONAL INSTITUTE OF STANDARDS &
TECHNOLOGY

Research Information Center
Gaithersburg, MD 20899

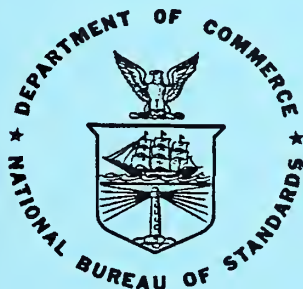
NBSIR 88-3830

The ICST-NBS Information Resource Dictionary System Command Language Prototype

Alan Goldfine
Thomasin Kirkendall

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

August 1988



75 Years Stimulating America's Progress
1913-1988

U.S. DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS

NBSIR 88-3830

**THE ICST-NBS INFORMATION RESOURCE
DICTIONARY SYSTEM COMMAND LANGUAGE
PROTOTYPE**

Alan Goldfine
Thomasin Kirkendall

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Institute for Computer Sciences and Technology
Gaithersburg, MD 20899

August 1988

U.S. DEPARTMENT OF COMMERCE, C. William Verity, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

THE ICST-NBS INFORMATION RESOURCE DICTIONARY SYSTEM
COMMAND LANGUAGE PROTOTYPE

Alan Goldfine
Thomasin Kirkendall

This publication is a report on the Information Resource Dictionary System (IRDS) Command Language prototype developed by the Institute for Computer Sciences and Technology of the National Bureau of Standards. It discusses the structure, source code, and operating environment of the IRDS Prototype, specifies the precise subset of the standard IRDS Command Language implemented, provides instructions for installing the Prototype software, and leads the reader through a typical user session.

Key words: command language; data dictionary; data dictionary system; Information Resource Dictionary System; IRDS; prototype.

ACKNOWLEDGMENTS

We would like to gratefully acknowledge the summer trainees and coop students, working at ICST-NBS, who contributed to the programming, installation, testing, and documentation of the Prototype: Sam Cook, Joe Farrington, Jim Gould, Richard Morris, and Doug White.

TABLE OF CONTENTS

	<u>Page</u>
1. AN OVERVIEW OF THE IRDS PROTOTYPE	1
1.1 HISTORY	1
1.2 OPERATING ENVIRONMENT	1
1.3 DISTRIBUTION OF THE IRDS PROTOTYPE	2
1.4 SCOPE AND USE OF THIS REPORT	3
2. AN IRDS PROTOTYPE SESSION	4
3. THE IRDS PROTOTYPE COMMAND LANGUAGE	6
3.1 NOTATION	7
3.2 IRD COMMANDS	7
3.2.1 ADD ENTITY	7
3.2.2 MODIFY ENTITY	8
3.2.3 DELETE ENTITY	9
3.2.4 ADD RELATIONSHIP	9
3.2.5 MODIFY RELATIONSHIP	10
3.2.6 DELETE RELATIONSHIP	11
3.2.7 MODIFY ENTITY ACCESS-NAME	11
3.2.8 MODIFY ENTITY DESCRIPTIVE-NAME	12
3.2.9 COPY ENTITY	12
3.2.10 GENERAL OUTPUT	13
3.3 IRD-SCHEMA COMMANDS	15
3.3.1 ADD META-ENTITY	15
3.3.2 MODIFY META-ENTITY	16
3.3.3 DELETE META-ENTITY	16
3.3.4 ADD META-RELATIONSHIP	17
3.3.5 MODIFY META-RELATIONSHIP	17
3.3.6 DELETE META-RELATIONSHIP	18
3.3.7 MODIFY META-ENTITY ACCESS-NAME	18
3.3.8 OUTPUT IRD-SCHEMA	19
3.4 UTILITY COMMANDS	21
3.4.1 CREATE IRD	21
3.4.2 REMOVE IRD	22
3.4.3 EXIT	23
3.4.4 HELP	23
3.5 ERROR MESSAGES	23
3.6 COMMAND LANGUAGE ABBREVIATIONS	24
4. THE IRDS PROTOTYPE SCHEMA	25
4.1 THE STRUCTURE OF THE SQL TABLES	25
4.1.1 The META-ATTRIBUTE-TYPE Table	25
4.1.2 The META-ATTRIBUTE-GROUP-TYPE/META-ATTRIBUTE- TYPE Table	26

	<u>Page</u>
4.1.3 The META-ENTITY-TYPE/META-ATTRIBUTE-TYPE Table	26
4.1.4 The META-ENTITY-TYPE/META-ATTRIBUTE-GROUP-TYPE Table	28
4.1.5 The META-ENTITY/META-ATTRIBUTE Table	28
4.1.6 The META-ENTITY/META-ATTRIBUTE-GROUP Table . .	30
4.1.7 The META-RELATIONSHIP-TYPE/META-ATTRIBUTE-TYPE Table	30
4.1.8 The META-RELATIONSHIP/META-ATTRIBUTE Table . .	31
4.1.9 The ENTITY/ATTRIBUTE Table	32
4.1.10 The ENTITY/ATTRIBUTE-GROUP Table	33
4.1.11 The RELATIONSHIP/ATTRIBUTE Table	33
4.2 IMPLEMENTOR DEFINED VALUES IN THE IRDS PROTOTYPE	34
4.2.1 Values For Meta-Attribute-Types	34
4.2.2 Values For Meta-Entities	36
5. THE IRDS PROTOTYPE SOURCE CODE	42
5.1 OVERVIEW	42
5.2 DICTIONARY SUBROUTINES	42
5.3 PARSING THE COMMANDS	43
5.4 COMMAND SUBROUTINES	44
5.5 OCI SUBROUTINES	44
5.6 HLI SUBROUTINES	44
5.7 GLOBAL VARIABLES	45
5.8 PROGRAM DATA STRUCTURES	45
6. INSTALLATION INSTRUCTIONS	47

1. AN OVERVIEW OF THE IRDS PROTOTYPE

1.1 HISTORY

Specifications for the Information Resource Dictionary System (IRDS), the emerging standard for data dictionary software, have been under development since 1980 as a joint effort of the Institute for Computer Sciences and Technology of the National Bureau of Standards (ICST-NBS) and Technical Committee H4 of the Accredited Standards Committee X3 (X3H4) [1].

Because the IRDS specifications, in particular those for the IRDS Command Language, describe a system quite different from currently available commercial data dictionary systems, ICST-NBS decided to develop a prototype Command Language implementation. The initial goal was to produce an IRDS prototype that would serve as a tool allowing experimentation on, and testing of, both the overall IRDS capabilities and the particular Command Language syntax. Later, this IRDS prototype would be available for use by organizations wishing to become familiar with the upcoming standard.

The IRDS Prototype was developed and used for testing the Specifications during 1985-1986. This coincided with the period of public and Federal Government agency review of the IRDS. In 1986, ICST-NBS began distributing the IRDS Prototype source code to interested outside organizations. In 1988, ICST-NBS released a revised version of the IRDS Prototype that is compatible with the final, standard specifications.

1.2 OPERATING ENVIRONMENT

The Prototype uses SQL calls to the ORACLE¹ database management system to model the IRDS data structures and to provide the underlying data management. A set of C language programs interpret the Prototype commands and interface with the DBMS.

ORACLE was chosen as the DBMS because it was available, was appropriate for the task, and because it implemented the

¹ ORACLE is a registered trademark of Oracle Corporation.

SQL standard. This use, however, should not be considered an endorsement or certification of the ORACLE product.

The Prototype is designed to be independent of the particular hardware environment and operating system of the system hosting the C compiler and Oracle DBMS.

1.3 DISTRIBUTION OF THE IRDS PROTOTYPE

The source code for the Prototype is available free of charge to interested organizations. The code is distributed on 5 1/4 inch, double-sided, double-density diskettes, stored in ASCII text file format. The files are readable by any computer using the DOS operating system.

ICST-NBS is distributing the Prototype to allow organizations to experiment with the emerging IRDS Standard Command Language. Users are encouraged to evaluate the Prototype software, and the underlying IRDS Specifications, for correctness, design philosophy, and desirable enhancements. Users are also asked to provide ICST-NBS with feedback concerning their experiences with the Prototype.

Users of the IRDS Prototype must agree to fully identify and credit ICST-NBS as the developer of the Prototype in any publications, talks, reports, or products that are based on work utilizing the Prototype.

The ICST-NBS IRDS Prototype is in the public domain, and no restrictions are placed on its use. It is not subject to copyright in the United States. ICST-NBS provides no warranty, and is exempt of any liability.

To find out more about the IRDS Prototype, or to request a copy of the source code, please contact:

IRDS Prototype Project
National Bureau of Standards
Information Systems Engineering Division
Building 225, Room A266
Gaithersburg, MD 20899

Tel: (301)975-3252

1.4 SCOPE AND USE OF THIS REPORT

The remainder of this report begins with a detailed depiction of a typical IRDS Prototype session, including a discussion of how to create new dictionaries. Chapter 3 follows with a description of the Prototype Command Language, including a description of the available commands, clauses, error messages, and allowable abbreviations. Chapter 4 discusses the structure of the SQL tables that store the IRD data and "implementor defined" parameter values used by the Prototype. In Chapter 5, the source code that implements the Prototype user interface is discussed. Much of the material in Chapters 4 and 5 may be of interest primarily to dictionary administrators. Finally, Chapter 6 presents a detailed set of instructions for installing the Prototype software.

This report deals only with the ICST-NBS Prototype. It does not provide a complete description of the IRDS Standard, the details of the Command Language, or any guidelines on IRDS usage. We recommend that users read the IRDS Technical Overview [2] as a tutorial and a general reference. A discussion, with many examples, of the complete Command Language is found in [3]. Guidelines for IRDS applications are presented in [4], and a guide on data entity naming conventions, within the framework of the IRDS, can be found in [5].

2. AN IRDS PROTOTYPE SESSION

Once the Prototype software has been installed, according to directions in Chapter 6, a user accesses the Prototype by running the executable file.

A session begins with the display of some package information giving the Prototype version number and the date that version was compiled. This is followed by the request:

IRDS_user_name :

The Prototype has no facilities for validating the user name that is entered; the information is used exclusively for audit purposes, such as for ADDED-BY attributes.

The Prototype then asks:

Is this a batch or interactive run (b/i)?

If the user enters "b", each user command is echoed, so the command string itself will be recorded as part of the batch output copy. An "i" specifies no echoing of the command string, and so is the normal response for a user working at a terminal.

Since each copy of the Prototype can support 25 discrete dictionaries, the Prototype will, in general, display at this point a menu of all previously created IRDs:

Available IRDs are:

- a) <name of first IRD>
 - b) <name of second IRD>
 - c) <name of third IRD>
 - .
 - .
 - .

Please specify your choice (letter)

The user must select one of the specified choices, even if he or she intends to create a new IRD.

The Prototype acknowledges the selection with

The current IRD is <name of IRD>

The Prototype then places the user "in" the selected IRD, and returns the prompt symbol ">". If the selected IRD is the one desired, the user can now begin working. If, on the other hand, the user wishes to create a new IRD, he or she does so at this point, using CREATE IRD (see section 3.3.1).

If there are no previously created IRDs to select from, the Prototype will not display the above menu of existing IRDs, but will generate an implicit CREATE IRD command, and display the following:

```

INFORMATION IXXXX: Creating 1st schema table
INFORMATION IXXXX: Creating 2nd schema table
INFORMATION IXXXX: Creating 3rd schema table
INFORMATION IXXXX: Creating 1st data table
INFORMATION IXXXX: Creating 2nd data table
INFORMATION IXXXX: Creating 3rd data table
INFORMATION IXXXX: All done.

```

The current IRD has no name.
What name do you want to give it?

The Prototype names the new IRD, displays

The current IRD is <name of IRD>

places the user in this IRD, and returns the prompt symbol ">".

It should be emphasized that the IRDS Command Language requires the use of the semicolon as the terminator of a command. The Prototype will take no action, and will remain in a wait state if the user forgets to place a semicolon at the end of a command.

3. THE IRDS PROTOTYPE COMMAND LANGUAGE

The IRDS Prototype currently implements the following 21 commands:

IRD Commands

```
ADD ENTITY
MODIFY ENTITY
DELETE ENTITY
ADD RELATIONSHIP
MODIFY RELATIONSHIP
DELETE RELATIONSHIP
MODIFY ENTITY ACCESS-NAME
MODIFY ENTITY DESCRIPTIVE-NAME
COPY ENTITY
OUTPUT IRD
```

IRD-Schema Commands

```
ADD META-ENTITY
MODIFY META-ENTITY
DELETE META-ENTITY
ADD META-RELATIONSHIP
MODIFY META-RELATIONSHIP
DELETE META-RELATIONSHIP
MODIFY META-ENTITY ACCESS-NAME
OUTPUT IRD-SCHEMA
```

Utility Commands

```
CREATE IRD
REMOVE IRD
EXIT
HELP
```

The HELP facility, in addition to providing users with on-line assistance, also serves to document the precise subset of the IRDS Command Language implemented in the current version of the Prototype.

The following sections present, for each implemented command, the subset of the Command Language syntax that has been included in the Prototype, along with one or more examples of the command's use. The format of the Prototype's response to a correctly specified command is also described, as are any differences between the Prototype implementation

and the Standard Command Language, as defined in the IRDS Specifications [1], and discussed in [2] and [3].

3.1 NOTATION

The construct $\left\{ \begin{array}{l} \text{A} \\ \text{B} \end{array} \right.$ in the syntax listings below

represents a choice between the clauses A and B.

Words in capitals, such as ADD, ENTITY, and DESCRIPTIVE-NAME, are IRDS-defined words.

Angle brackets "<" and ">" enclose syntactic categories, e.g., "<access-name>" and "<attribute-clause>".

Square brackets "[" and "]" enclose optional items. A string of the form "[, <C> ...]" represents the occurrence of zero or more instances of syntactic category C.

3.2 IRD COMMANDS

3.2.1 ADD ENTITY

Syntax:

```
ADD ENTITY <access-name> ENTITY-TYPE = <entity-type>
  [ ENTITY DESCRIPTIVE-NAME = <descriptive-name>]
  [ WITH [ATTRIBUTES] <attribute-clause>
    [ , <attribute-clause> ... ] ] ;
```

where <attribute-clause> is:

```
<attribute-type> = <attribute>
or
<attribute-group-type> =
  (
    <component-attribute-type> = <attribute>
    [ , <component-attribute-type> = <attribute>
    ... ] )
```

Examples:

```
add entity u8 entity-type = system;

add entity u8 entity-type = system
  entity descriptive-name = example_system
```

```
with comments = "this is an example system";

add entity u8 entity-type = system
  with external-security = "none",
       location = "example book",
       identification-names =
         (alternate-name = "example",
          alternate-name-context = "here");
```

Prototype Response:

Entity <access-name> added.

3.2.2 MODIFY ENTITY

Syntax:

```
MODIFY ENTITY <access-name>
  [ ENTITY DESCRIPTIVE-NAME = <descriptive-name> ]
  [ WITH [ ATTRIBUTES ] <attribute-clause>
        [, <attribute-clause> ...] ;

where <attribute-clause> is
  <attribute-type> = <attribute>
    or
  <attribute-group-type> =
    (
      <component-attribute-type> = <attribute>
      [, <component-attribute-type> = <attribute> ... ] )
```

Examples:

```
modify entity PAYROLL-SYSTEM with
  external-security = "confidential",
  identification-names =
    (alternate-name = "PAYROLL-SYS",
     alternate-name-context = "DIVISION-100");

modify entity AS
  entity descriptive-name = ACCOUNTING-SYSTEM;
```


Prototype Response:

Entity <access-name> modified.

3.2.3 DELETE ENTITYSyntax:

```
DELETE ENTITY <access-name> [ , access-name ... ] ;
```

Examples:

```
delete entity u8a-30;
```

```
delete entity u8a-30, u8a-31, u8a-32;
```

Prototype Response:

Entity <access-name> deleted. . . . Entity <access-name> deleted.

3.2.4 ADD RELATIONSHIPSyntax:

```
ADD RELATIONSHIP
```

```
<access-name-1> { /<relationship-type>  
                  \<relationship-class-type>
```

```
[ NEW [ <entity-2-type> ] ] <access-name-2>
```

```
[ WITH [ATTRIBUTES] <attribute-type> = <attribute>  
        [ , <attribute-type> = <attribute> ... ] ] ;
```

Examples:

```
add relationship u8 system-contains-system u8a;
add relationship u8 contains new system u8a-30;
add relationship u8 system-contains-system new u8a-30;
add relationship u-8 processes payroll
    with access-method = "protected", frequency =
                        "bi-monthly";
```

Prototype Response:

```
Relationship <access-name-1> <relationship-type>
    <access-name-2> added.
```

3.2.5 MODIFY RELATIONSHIP

Syntax:

MODIFY RELATIONSHIP

```
<access-name-1> { /<relationship-type>
                  \<relationship-class-type>
    <access-name-2>
    [ WITH [ATTRIBUTES] <attribute-type> = <attribute>
      [ , <attribute-type> = <attribute> ... ] ] ;
```

Example:

```
modify relationship u8 processes payroll with
    frequency = "50", access-method = "direct";
```

Prototype Response:

```
Relationship <access-name-1> <relationship-type>
    <access-name-2> modified.
```

3.2.6 DELETE RELATIONSHIP

Syntax:

DELETE RELATIONSHIP

```

<access-name-1> { / <relationship-type>
                  \ <relationship-class-type>
                  <access-name-2>

[ , <access-name-1> { / <relationship-type>
                    \ <relationship-class-type>
                    <access-name-2>

                    ... ] ;

```

Examples:

delete relationship u8 system-contains-system u8-30;

delete relationship u8 contains u8-25, u8 contains u8a;

Prototype Response:

```

Relationship <access-name-1> <relationship-type>
                        <access-name-2> deleted.
      .
      .
      .
Relationship <access-name-1> <relationship-type>
                        <access-name-2> deleted.

```

3.2.7 MODIFY ENTITY ACCESS-NAME

Syntax:

MODIFY ENTITY ACCESS-NAME FROM <old-name> TO <new-name> ;

Example:

```
modify entity access-name from u8-20 to test1;
```

Prototype Response:

```
Entity access-name modified from <old-name> to
                                         <new-name>.
```

3.2.8 MODIFY ENTITY DESCRIPTIVE-NAME

Syntax:

```
MODIFY ENTITY DESCRIPTIVE-NAME FROM <old-name>
                                TO <new-name> ;
```

Example:

```
modify entity descriptive-name from
    Old-Long-Name-1234567890 to New-Long-Name-1234567890;
```

Prototype Response:

Entity descriptive-name modified from <old-name> to
 <new-name> for <access-name>.

3.2.9 COPY ENTITY

Syntax:

```

COPY ENTITY <access-name-1> [ WITH RELATIONSHIPS ]
    TO <access-name-2>
    [ ENTITY DESCRIPTIVE-NAME = <descriptive-name> ] ;

```

Examples:

```
copy entity u8-20-10 with relationships to New-u8-20-10;
```

```
copy entity Tape_recording to Memoirs
    entity descriptive-name = Life and Times;
```


Prototype Response:

Entity <access-name-1> copied to
entity <access-name-2>.

3.2.10 GENERAL OUTPUTSyntax:

OUTPUT IRD

```

SELECT {
  ALL ENTITIES
  ENTITIES [WITH] ACCESS-NAME = <scan-mask>
                                     [, <scan-mask> ... ]
  ENTITIES [WITH] DESCRIPTIVE-NAME =
                                     scan-mask [, <scan-mask> ... ]
  ENTITIES DIRECTLY RELATED TO <access-name>
                                     [, <access-name> ... ]
}

[WHERE
  < conditional expression using "(", ")", "AND", "OR"
  and subexpressions:
    ENTITY-TYPE = <entity-type> [, <entity-type>...]
    ENTITY ASSIGNED ACCESS-NAME <rel-op>
                                     <access-name>
    ENTITY ASSIGNED DESCRIPTIVE-NAME <rel-op>
                                     <descriptive-name>
    <attribute-type> <rel-op> <attribute> > ]

SHOW {
  ALL
  ENTITY ACCESS-NAME
  ENTITY DESCRIPTIVE-NAME
  ALL ATTRIBUTES
  ALL RELATIONSHIPS
}

;
```

A <scan-mask> may use, in addition to explicitly specified characters, the substitution characters "*" and "?". Substitution character "*" matches any sequence of characters, including the null sequence; substitution character "?" matches any single character other than null. The term <rel-op> refers to one of the operators:

"=", equal to,
"/=", not equal to,
">", greater than,
"<", less than,
">=" or "<=", greater than or equal to,
"<=" or ">", less than or equal to.

Examples:

```
output ird select all show all;
```

```
output ird
  select entities with access-name = *database*, dbms*
    where entity-type = file
  show entity access-name;
```

Prototype Response:

For each of the entities in a hypothetical IRD reported on by OUTPUT IRD SELECT ALL SHOW ALL; the Prototype would generate a display looking something like:

Entity = H-I

Descriptive-Name = Health-Insurance

Entity-Type = SYSTEM

Attributes

Added-By = Goldfine

Last-Modified-By = Kirk

o

o

o

System-Category = Personnel

Attribute-Groups

Date-Time-Added

System-Date = 19860720

System-Time = 152654

Date-Time-Last-Modified

System-Date = 19860723

System-Time = 093150

Relationships

```

H-I SYSTEM-PROCESSES-FILE H-I-Carrier
  ACCESS-METHOD = Direct
  FREQUENCY = Weekly
J_Smith USER-RUNS-SYSTEM H-I
  FREQUENCY = Daily
      o
      o
      o

```

At the end of the output, the following message is displayed:

```

IRD output completed.

```

3.3 IRD-SCHEMA COMMANDS

3.3.1 ADD META-ENTITY

Syntax:

```

ADD META-ENTITY <meta-entity-name> META-ENTITY-TYPE =
                                     <meta-entity-type>
[ WITH [META-ATTRIBUTES]
  <meta-attribute-type> = <meta-attribute>
  [, <meta-attribute-type> = <meta-attribute> ... ] ] ;

```

Examples:

```

add meta-entity COLOR meta-entity-type = attribute-type;

add meta-entity COLOR meta-entity-type = attribute-type
  with purpose = "this is attribute-type is used to
define the color of a DOCUMENT";

```

Prototype Response:

```

Meta-entity <meta-entity-name> added.

```

3.3.2 MODIFY META-ENTITY

Syntax:

```
MODIFY META-ENTITY <meta-entity-name>  
  WITH [META-ATTRIBUTES] <attribute-type> = <attribute>  
      [, <attribute-type> = <attribute> ... ] ;
```

Examples:

```
modify meta-entity PUBLICATION  
  with purpose = "this entity-type refers only to formally  
published documents" ;
```

```
modify meta-entity COLOR  
  with maximum-number-of-occurrences = 7,  
      format = string ;
```

Prototype Response:

Meta-entity <meta-entity-name> modified.

3.3.3 DELETE META-ENTITY

Syntax:

```
DELETE META-ENTITY <meta-entity-name>  
  [WITH META-RELATIONSHIPS] ;
```

Example:

```
delete meta-entity u8a-30;
```

Prototype Response:

Meta-entity <meta-entity-name> deleted.

3.3.4 ADD META-RELATIONSHIP

Syntax:

ADD META-RELATIONSHIP

```

<meta-entity-1> { /<meta-relationship-type>
                  \ <meta-relationship-class-type>

<meta-entity-2> [ POSITION = <n> ]

[WITH [META-ATTRIBUTES]
  <meta-attribute-type> = <meta-attribute>
[, <meta-attribute-type> = <meta-attribute> ... ] ] ;

```

Example:

```

add meta-relationship
  document-contains-program connects document
  position = 1 with purpose = "example";

```

Prototype Response:

```

Meta-relationship <meta-entity-1>
  <meta-relationship-type> <meta-entity-2> added.

```

3.3.5 MODIFY META-RELATIONSHIP

Syntax:

MODIFY META-RELATIONSHIP

```

<meta-entity-1> { /<meta-relationship-type>
                  \ <meta-relationship-class-type>

<meta-entity-2> [ POSITION = <n> ]

[WITH [META-ATTRIBUTES]
  <meta-attribute-type> = <meta-attribute>
[, <meta-attribute-type> = <meta-attribute> ... ] ] ;

```

Example:

```
modify meta-relationship
  document-contains-program connects program position = 2
  with purpose = "another example";
```

Prototype Response:

Meta-relationship <meta-entity-1> <meta-relationship-type> <meta-entity-2> modified.

3.3.6 DELETE META-RELATIONSHIP

Syntax:

DELETE META-RELATIONSHIP

```
<meta-entity-1> { /<meta-relationship-type>
                  \ <meta-relationship-class-type>
  <meta-entity-2> [ POSITION = <n> ] ;
```

Example:

```
delete meta-relationship
  document-contains-program connects program
  position = 2;
```

Prototype Response:

Meta-relationship <meta-entity-1> <meta-relationship-type> <meta-entity-2> deleted.
--

3.3.7 MODIFY META-ENTITY ACCESS-NAME

Syntax:

```
MODIFY META-ENTITY ACCESS-NAME
  FROM <meta-entity-access-name-1>
  TO   <meta-entity-access-name-2> ;
```

Example:

modify meta-entity access-name from document to report;

Prototype Response:

Meta-entity access-name modified from
 <meta-entity-access-name-1>
 to <meta-entity-access-name-2>.

3.3.8 OUTPUT IRD-SCHEMASyntax:

OUTPUT IRD-SCHEMA

```

SELECT { / ALL
          <meta-entity-name>
          [, <meta-entity-name> ... ]

SHOW { / ALL
        ALL META-ATTRIBUTES
        META-ATTRIBUTES
        <meta-attribute-type>
        [, <meta-attribute-type> ... ]

```

;

Example:

output ird-schema select document show all;

Prototype Response:

For this command, the Prototype would generate a display looking something like:

Meta-Entity = DOCUMENT
 Meta-Entity-Type = ENTITY-TYPE

Meta-Attributes

```
Added-By = BASIC-FUNCTIONAL-SCHEMA
Meta-Entity-Substitute-Name = DOC
Connectable = YES
    o
    o
    o
System-Generated = NO
System-Lock = ON
```

Meta-Attribute-Groups

```
DATE-TIME-ADDED
    SYSTEM-DATE = 19860720
    SYSTEM-TIME = 152654
DATE-TIME-LAST-MODIFIED
    SYSTEM-DATE = 19860723
    SYSTEM-TIME = 093150
```

Meta-Relationships

```
DOCUMENT ENTITY-TYPE-CONTAINS-ATTRIBUTE-TYPE
                                         ADDED-BY
    Implementation-Lock = OFF
        o
        o
        o
    System-Lock = ON
DOCUMENT ENTITY-TYPE-CONTAINS-ATTRIBUTE-TYPE
                                         CLASSIFICATION
    Implementation-Lock = OFF
        o
        o
        o
    System-Lock = OFF
        o
        o
        o
DOCUMENT ENTITY-TYPE-CONTAINS-ATTRIBUTE-GROUP-TYPE
                                         IDENTIFICATION-NAMES
    Implementation-Lock = OFF
        o
        o
        o
```



```

      System-Lock = OFF
DOCUMENT-CONTAINS-DOCUMENT
      RELATIONSHIP-TYPE-CONNECTS-ENTITY-TYPE
                                          DOCUMENT
      Implementation-Lock = OFF
          o
          o
          o
      System-Lock = OFF
          o
          o
          o
USER-RESPONSIBLE-FOR-DOCUMENT
      RELATIONSHIP-TYPE-CONNECTS-ENTITY-TYPE
                                          DOCUMENT
      Implementation-Lock = OFF
          o
          o
          o
      System-Lock = OFF

```

At the end of the output, the following message is displayed:

```

IRD-SCHEMA output completed.

```

NOTE: Care should be taken in issuing the command:

```

output ird-schema select all show all;

```

This command will cause the display of the entire IRD-Schema, which will include the Minimal Schema and, unless it has been redefined, the Basic Functional Schema. Over 350,000 characters of text are generated in the display of the Minimal and Basic Functional Schemas.

3.4 UTILITY COMMANDS

3.4.1 CREATE IRD

Syntax:

```

CREATE IRD <IRD-name> IRD-SCHEMA IS STANDARD ;

```

Example:

```
create ird production-2 ird-schema is standard;
```

The term "standard" in the Prototype's CREATE IRD command refers to a combination of the Minimal Schema and the Basic Functional Schema of the IRDS Standard.

Prototype Response:

```
INFORMATION IXXXX: Creating 1st schema table
INFORMATION IXXXX: Creating 2nd schema table
INFORMATION IXXXX: Creating 3rd schema table
INFORMATION IXXXX: Creating 1st data table
INFORMATION IXXXX: Creating 2nd data table
INFORMATION IXXXX: Creating 3rd data table
INFORMATION IXXXX: All done.
```

3.4.2 REMOVE IRD

Syntax:

```
REMOVE IRD <IRD-name> ;
```

Example:

```
remove ird test-04;
```

Prototype Response:

```
IRD <IRD-name> removed.
```

The Specifications for the IRDS Command Language do not contain a REMOVE IRD command. However, the ability to create new IRDs certainly implies the need to remove them. Hence the Prototype was implemented with this command.

3.4.3 EXIT

Syntax:

EXIT ;

Example:

exit;

Prototype Response:

Return to calling program or operating system.

3.4.4 HELP

Syntax:

HELP [<command>] ;

Examples:

help;

help add meta-relationship;

Prototype Response:

For HELP;, a list of the currently available commands.

For HELP <command>;, a description of the syntax of that command, and some examples of command usage.

3.5 ERROR MESSAGES

The Prototype generates all the appropriate error messages specified in the IRDS Standard. In addition, certain error conditions that are not documented in the Specifications are recognized by the Prototype. These conditions cause the generation of self explanatory error messages beginning with "EXXXXX:".

3.6 COMMAND LANGUAGE ABBREVIATIONS

The Prototype accepts abbreviations for a set of IRDS-words that are defined in the Standard and that are part of the Command Language. An abbreviation can be used anywhere in place of its corresponding full formulation.

<u>IRDS-word</u>	<u>Abbreviation</u>
ACCESS-NAME	NAME
ALTERNATE-NAME	ANAME
ASSIGNED	ASSGN
ATTRIBUTES	ATTRB
ATTRIBUTE-TYPE	ATYPE
COPY	CPY
CREATE	CRE
DELETE	DEL
DESCRIPTIVE-NAME	DNAME
ENTITY-TYPE	ETYPE
META-ATTRIBUTES	MATRBS
META-ENTITY	MENTY
META-ENTITY-TYPE	METYPE
META-RELATIONSHIP	MREL
META-RELATIONSHIPS	MRELS
MODIFY	MOD
OUTPUT	OUT
RELATIONSHIP	REL
RELATIONSHIPS	RELS
RELATIONSHIP-TYPE	RTYPE

The Prototype also accepts the set of meta-entity substitute-names, such as DOC for DOCUMENT and SYS-CON-SYS for SYSTEM-CONTAINS-SYSTEM, defined as part of the "standard" schema. Appendices A and B of the IRDS Technical Overview [2] contain a complete list of these substitute-names.

4. THE IRDS PROTOTYPE SCHEMA

4.1 THE STRUCTURE OF THE SQL TABLES

Each IRD has associated with it eleven SQL tables, which contain all the IRD and IRD-schema data for that dictionary. These tables are:

- META-ATTRIBUTE-TYPE
- META-ATTRIBUTE-GROUP/META-ATTRIBUTE-TYPE
- META-ENTITY-TYPE/META-ATTRIBUTE-TYPE
- META-ENTITY-TYPE/META-ATTRIBUTE-GROUP-TYPE
- META-ENTITY/META-ATTRIBUTE
- META-ENTITY/META-ATTRIBUTE-GROUP
- META-RELATIONSHIP-TYPE/META-ATTRIBUTE-TYPE
- META-RELATIONSHIP/META-ATTRIBUTE
- ENTITY/ATTRIBUTE
- ENTITY/ATTRIBUTE-GROUP
- RELATIONSHIP/ATTRIBUTE

The following sections present the SQL definitions for each of these tables.

4.1.1 The META-ATTRIBUTE-TYPE Table

The META-ATTRIBUTE-TYPE table (MATYPE) stores the descriptive information defining the Prototype's meta-attribute-types, as specified in section 9.3 of Module 1 of the IRDS Specifications [1]. Each row of the table corresponds to a meta-attribute-type; the columns could be said to correspond to meta-meta-attribute-types. Once the Prototype source code is compiled, MATYPE is fixed, in that there is no provision in the Standard for a user to be able to redefine meta-attribute-types. Since it is fixed, MATYPE is stored once, and is shared by all IRDs using the given executable.

Definition:

```
create table MATYPE
```

(meta_attribute_type_name	char (65),
internal_name	char (30),
description	char (240),
format	char (7),
minimum_length	integer (2),

maximum_length	integer (5),
default_	char (20),
constraints	char (240),
repeating	char (3),
system_maintained	char (3),
fixed	char (3),
required	char (3),
uniqueness_rules	char (3))

4.1.2 The META-ATTRIBUTE-GROUP-TYPE/META-ATTRIBUTE-TYPE Table

The META-ATTRIBUTE-GROUP-TYPE/META-ATTRIBUTE-TYPE (MAGTYPE_MATYPE) table describes the association between the meta-attribute-group-types and their component meta-attribute-types in the Prototype's IRD-schema, as specified in section 9.6 and Table 3 of Module 1 of the IRDS Specifications. Each row of the table corresponds to a component meta-attribute-type of a meta-attribute-group-type; each column corresponds to a meta-attribute-type. MAGTYPE_MATYPE is fixed, stored once, and shared by all IRDs.

Definition:

create table MAGTYPE_MATYPE

(magtype	char (64),
internal_name	char (30),
matype	char (64),
pos	integer (2),
sys_chars	char (2))

4.1.3 The META-ENTITY-TYPE/META-ATTRIBUTE-TYPE Table

The META-ENTITY-TYPE/META-ATTRIBUTE-TYPE (METYPE_MATYPE) table describes the correspondence between the meta-entity-types and their associated meta-attribute-types in the Prototype's IRD-schema, as specified in section 9.4 and Table 1 of Module 1 of the IRDS Specifications. Each row of the table corresponds to a meta-entity-type; each column corresponds to a meta-attribute-type. METYPE_MATYPE is fixed, stored once, and shared by all IRDs.

Definition:

```

create table METYPE_MATYPE
(me_type          char (40),
defined_by       char (2),
alt_mname        char (1),
common           char (2),
connectable      char (1),
e_class          char (1),
fmt              char (2),
i_lock           char (1),
integer_limit    char (3),
inverse          char (1),
last_changed_by  char (1),
origin           char (3),
phase_class      char (2),
line_count_limit char (3),
line_length_limit char (3),
max_lngth        char (2),
max_dname_lngth  char (2),
max_dname_lngth_def char (1),
max_name_lngth   char (2),
max_name_lngth_def char (1),
max_name_limit   char (3),
max_occ_def      char (1),
max_occ_limit    char (3),
min_lngth        char (2),
min_dname_lngth  char (2),
min_dname_lngth_def char (1),
min_name_lngth   char (2),
min_name_lngth_def char (1),
i_count          char (2),
mod_count        char (1),
pic              char (1),
purpose          char (1),
seq              char (2),
sig_attrbs       char (2),
st_name          char (1),
string_length_limit char (3),
sys_gened        char (2),
sys_lock         char (2),
validation_type  char (1),
var_lngth_limit  char (3),
rule_desc        char (1),
max_dname_lngth_lim char (3),
max_menty_ass_name_lim char (3),
max_menty_ass_dname_lim char (3),
r_name          char (4),

```


r_mname	char (4),
mode	char (1),
sys_maint	char (1),
grp_txt_alwd	char (3),
var	char (1))

4.1.4 The META-ENTITY-TYPE/META-ATTRIBUTE-GROUP-TYPE Table

The META-ENTITY-TYPE/META-ATTRIBUTE-GROUP-TYPE (METYPE_MAGTYPE) table describes the correspondence between the meta-entity-types and their associated meta-attribute-group-types in the Prototype's IRD-schema, as specified by section 9.7 and Table 4 of Module 1 of the IRDS Specifications. Each row corresponds to a meta-entity-type; each column corresponds to a component meta-attribute-type of a meta-attribute-group-type. METYPE-MAGTYPE is fixed, stored once, and shared by all IRDs.

Definition:

```
create table METYPE_MAGTYPE
(metype                                char (64),
 data_range                           char (1),
 data_value                           char (1),
 added                                char (2),
 modified                             char (2) )
```

4.1.5 The META-ENTITY/META-ATTRIBUTE Table

The META-ENTITY/META-ATTRIBUTE (MENTY_MATT) table stores all meta-attributes associated with all meta-entities in the Prototype's IRD-schema. Each row corresponds to a meta-entity; each column corresponds to a meta-attribute-type. When a new IRD is created, the table is initially populated with the meta-entities in the Minimal Schema and the Basic Functional Schema, as specified in section 10.2.1 of Module 1 and section 5.1 of Module 2 of the IRDS Specifications. As new meta-entities are added to the IRD-schema, they are entered into this table.

Definition:

```
create table MENTY_MATT
(me_type                                char (35),
 menty                                 char (64),
```


id_number	integer (3),
internal_name	char (30),
menty_variation_name	char (8),
menty_revision_number	integer (1),
menty_ass_dname	char (64),
defined_by	char (32),
alt_mname	char (32),
common	char (3),
connectable	char (3),
e_class	char (8),
fmt	char (7),
i_lock	char (3),
integer_limit	integer (22),
inverse	char (64),
last_changed_by	char (32),
origin	char (8),
phase_class	char (12),
line_count_limit	integer (5),
line_length_limit	integer (3),
max_lngth	integer (5),
max_dname_lngth	integer (3),
max_dname_lngth_def	integer (3),
max_name_lngth	integer (3),
max_name_lngth_def	integer (3),
max_name_limit	integer (3),
max_occ_def	integer (3),
max_occ_limit	integer (3),
min_lngth	integer (2),
min_dname_lngth	integer (2),
min_dname_lngth_def	integer (2),
min_name_lngth	integer (1),
min_name_lngth_def	integer (1),
i_count	integer (9),
mod_count	integer (9),
pic	char (64),
purpose	char (65535),
seq	char (3),
sig_attrbs	integer (2),
st_name	char (31),
string_length_limit	integer (3),
sys_gened	char (3),
sys_lock	char (3),
validation_type	char (5),
var	char (31),
var_lngth_limit	integer (2),
rule_desc	char (1),
max_dname_lngth_lim	integer (2),
max_menty_ass_name_lim	integer (2),

max_menty_ass_dname_lim	integer (2),
r_name	char (1),
r_mname	char (1),
mode_	char (8),
sys_maint	char (3),
grp_txt_alwd	char (3))

4.1.6 The META-ENTITY/META-ATTRIBUTE-GROUP Table

The META-ENTITY/META-ATTRIBUTE-GROUP (MENTY_MAG) table stores all meta-attribute-groups associated with all meta-entities in the Prototype's IRD-schema. Each row corresponds to a meta-entity; each column corresponds to a component meta-attribute-type of a meta-attribute-group-type. When a new IRD is created, the table is initially populated with the meta-entities in the Minimal Schema and the Basic Functional Schema, as specified in section 10.2.1 of Module 1 and section 5.1 of Module 2 of the IRDS Specifications. As new meta-entities are added to the IRD-schema, they are entered into this table.

Definition:

```
create table MENTY_MAG
(menty                                     char (64),
menty_var_name                           char (8),
menty_rev_num                            integer,
added$date                               char (8),
added$time                               char (6),
modified$date                            char (8),
modified$time                            char (6) )
```

4.1.7 The META-RELATIONSHIP-TYPE/META-ATTRIBUTE-TYPE Table

The META-RELATIONSHIP-TYPE/META-ATTRIBUTE-TYPE (MRTYPE_MATYPE) table describes the correspondence between the meta-relationship-types and their associated meta-attribute-types in the Prototype's IRD-schema, as specified in section 9.5 and Table 2 of Module 1 of the IRDS Specifications. Each row corresponds to a meta-relationship-type; each column corresponds to a meta-attribute-type. MRTYPE_MATYPE is fixed, stored once, and shared by all IRDs.

Definition:

```

create table MRTYPE_MATYPE
(mrtype                                integer (2),
 metype1                             char (35),
 metype2                             char (35),
 mrel_class_type                     char (9),
 mrel_type                           char (64),
 grp_pos                             char (2),
 i_lock                              char (2),
 max_occ                             char (1),
 origin                             char (3),
 pos                                 char (1),
 purpose                             char (1),
 seq_parm                            char (1),
 sing                                char (1),
 sys_lock                            char (2) )

```

4.1.8 The META-RELATIONSHIP/META-ATTRIBUTE Table

The META-RELATIONSHIP/META-ATTRIBUTE (MREL_MATT) table stores all meta-attributes associated with all meta-relationships in the Prototype's IRD-schema. Each row corresponds to a meta-relationship; each column corresponds to a meta-attribute-type. When a new IRD is created, the table is initially populated with the meta-relationships defined in the Minimal Schema and the Basic Functional Schema, as specified in section 10.3 of Module 1 and section 6 of Module 2 of the IRDS Specifications. As new meta-relationships are added to the IRD-schema, they are entered into this table.

Definition:

```

create table MREL_MATT
(mrtype                                integer (2),
 menty1                              char (64),
 menty1_var                           char (8),
 menty1_rev_num                       integer,
 menty2                              char (64),
 menty2_var                           char (8),
 menty2_rev_num                       integer,
 grp_pos                             integer (2),
 i_lock                              char (3),
 max_occ                             integer (3),
 origin                             char (8),
 pos                                 integer (1),

```

purpose	char (65535),
seq_parm	char (3),
sing	char (8),
sys_lock	char (3))

4.1.9 The ENTITY/ATTRIBUTE Table

The ENTITY/ATTRIBUTE (ENTY_ATT) table stores all attributes associated with all entities in the application IRD. Each row corresponds to an entity; each column corresponds to an attribute-type defined in the schema of the application IRD. The table is empty when the IRD is created. As entities are added to the IRD, they are entered into this table. When new attribute-types are defined in the schema, corresponding columns are added to the table, making the table dynamic with respect to columns as well as rows.

The following definition is not an extract from the Prototype source code, but is equivalent to that more dynamic definition:

Definition:

```
create table ENTY_ATT
(entity_type          char (64),
 entity_name         char (32),
 var_name            char (8),
 rev_num             integer,
 descriptive_name     char (64),
 added_by            char (32),
 allowable_value     char (32),
 classification       char (32),
 code_list_location  char (32),
 comments            char (240),
 data_class          char (32),
 data_type           char (16),
 description         char (5000),
 dict_partition_name char (32),
 document_category   char (32),
 external_security    char (32),
 internal_format      char (32),
 ird_schema_phase_name char (32),
 justification        char (5),
 last_modified_by     char (32),
 length              integer,
 location            char (32),
 mod_count            integer,
```


integer_of_records	integer,
num_lines_code	integer,
precision	integer (2),
record_category	char (32),
scale	integer (2),
system_category	char (32),
usage	char (32))

4.1.10 The ENTITY/ATTRIBUTE-GROUP Table

The ENTITY/ATTRIBUTE-GROUP (ENTY_AG) table stores all attribute-groups associated with all entities in the application IRD. Each row corresponds to an entity; each column corresponds to a component attribute-type of an attribute-group-type defined in the schema of the application IRD. The table is empty when the IRD is created. As entities are added to the IRD, they are entered into this table. When new attribute-group-types are defined in the schema, corresponding columns are added to the table, making the table dynamic with respect to columns as well as rows.

The following definition is equivalent to the definition found in the source code:

Definition:

```
create table ENTY_AG
(entity_name                char (32),
 var_name                  char (8),
 rev_num                   integer,
 alw_range$high_of_range   char (32),
 alw_range$low_of_range    char (32),
 duration$duration_type    char (32),
 duration$duration_value   char (22),
 d_t_added$system_date     char (8),
 d_t_added$system_time    char (6),
 d_t_mod$system_date       char (8),
 d_t_mod$system_time       char (6),
 id_names$alternate_name   char (32),
 id_names$alt_name_context char (32) )
```

4.1.11 The RELATIONSHIP/ATTRIBUTE Table

The RELATIONSHIP/ATTRIBUTE (REL_ATT) table stores all attributes associated with all relationships in the application IRD. Each row corresponds to a relationship; each col-

umn corresponds to an attribute-type defined in the schema of the application IRD. The table is empty when the IRD is created. As relationships are added to the IRD, they are entered into this table. When new attribute-types are defined in the schema, corresponding columns are added to the table, making the table dynamic with respect to columns as well as rows.

Definition:

```
create table REL_ATT
(relationship_type          char (64),
 entity1                   char (32),
 var_name1                 char (8),
 rev_num1                  integer,
 entity2                   char (32),
 var_name2                 char (8),
 rev_num2                  integer,
 relationship_class_type   char (64)
 relationship_type         char (64),
 entity1                   char (32),
 var_name1                 char (8),
 rev_num1                  integer,
 entity2                   char (2),
 var_name2                 char (8),
 rev_num2                  integer,
 relationship_class_type   char (64),
 access_method             char (32),
 default_view              char (3),
 frequency                 char (32),
 relative_position         integer (22) )
```

4.2 IMPLEMENTOR DEFINED VALUES IN THE IRDS PROTOTYPE

The IRDS Standard Specifications [1] characterize many of the meta-meta-attributes and meta-attributes in the above tables as "implementor defined" or "installation specified" when applied to specific meta-attribute-types or meta-entities. The following sections list the values used in the Prototype for these meta-meta-attributes and meta-attributes.

4.2.1 Values For Meta-Attribute-Types

ADDED-BY

Maximum Length = 32

DECODED-VALUE

Maximum Length = 32

ENCODED-VALUE

Maximum Length = 32

GROUP-POSITION

Maximum Length = 2

HIGH-VALUE

Maximum Length = 22

INTEGER-LIMIT

Maximum Length = 22

INVERSE-NAME

Maximum Length = 32

LAST-MODIFIED-BY

Maximum Length = 32

LINE-COUNT-LIMIT

Maximum Length = 5

LOW-VALUE

Maximum Length = 22

MAXIMUM-NUMBER-OF-OCCURRENCES

Maximum Length = 3

MAXIMUM-NUMBER-OF-OCCURRENCES-DEFAULT

Maximum Length = 3

MAXIMUM-NUMBER-OF-OCCURRENCES-LIMIT

Maximum Length = 3

META-ENTITY-SUBSTITUTE-NAME

Maximum Length = 32

MINIMUM-ATTRIBUTE-LENGTH

Maximum Length =

NUMBER-OF-INSTANCES

Maximum Length = 22

NUMBER-OF-TIMES-MODIFIED

Maximum Length = 22

ORIGIN

Minimum Length = 6

Maximum Length = 8

PICTURE

Maximum Length = 32

PURPOSE

Minimum Length = 1

Maximum Length = 5000

SEQUENCE-PARAMETER

Minimum Length = 2

Maximum Length = 3

SIGNIFICANT-ATTRIBUTES

Maximum Length = 2

START-NAME

Maximum Length = 8

VARIATION

Maximum Length = 2

VARIATION-LENGTH-LIMIT

Maximum Length = 2

4.2.2 Values For Meta-Entities

The following are the implementor defined meta-attributes for the "Standard IRD-Schema" meta-entities:

Each meta-entity has either MINIMAL-SCHEMA or BASIC-FUNCTIONAL-SCHEMA, as appropriate, as its Added-By meta-attribute.

Entity-Types

Each entity-type has for its Meta-Entity-Substitute-Name the value given in sections A.1 and B.1 of the IRDS Technical Overview [2].

For each entity-type:

Maximum-Entity-Assigned-Access-Name-Length = 32

Maximum-Entity-Assigned-Descriptive-Name-Length = 64

Minimum-Entity-Assigned-Access-Name-Length = 1
 Minimum-Entity-Assigned-Descriptive-Name-Length = 1

Relationship-Types and Relationship-Class-Types

Each relationship-type and relationship-class-type has for its Meta-Entity-Substitute-Name the value given in sections A.2 and B.2 of the IRDS Technical Overview.

Attribute-Types

ADDED-BY

Maximum-Attribute-Length = 32
 Minimum-Attribute-Length = 1

DEFAULT-VIEW

Meta-Entity-Substitute-Name = DEF-VIEW
 Maximum-Attribute-Length = 3
 Minimum-Attribute-Length = 2

IRD-PARTITION-NAME

Maximum-Attribute-Length = 32
 Minimum-Attribute-Length = 1

LAST-MODIFIED-BY

Meta-Entity-Substitute-Name = LAST-MOD-BY
 Maximum-Attribute-Length = 32
 Minimum-Attribute-Length = 1

NUMBER-OF-TIMES-MODIFIED

Meta-Entity-Substitute-Name = NO-TIMES-MOD
 Maximum-Attribute-Length = 22
 Minimum-Attribute-Length = 1

IRD-SCHEMA-PHASE-NAME

Meta-Entity-Substitute-Name = S-PH-NAME
 Maximum-Attribute-Length = 32
 Minimum-Attribute-Length = 1

SYSTEM-DATE

Maximum-Attribute-Length = 8
 Minimum-Attribute-Length = 8

SYSTEM-TIME

Maximum-Attribute-Length = 6
 Minimum-Attribute-Length = 6

ACCESS-METHOD

Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

ALLOWABLE-VALUE

Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

ALTERNATE-NAME

Meta-Entity-Substitute-Name = ALT-NAME
Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

ALTERNATE-NAME-CONTEXT

Meta-Entity-Substitute-Name = ALT-NAME-CONTEXT
Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

CLASSIFICATION

Meta-Entity-Substitute-Name = CLASS
Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

CODE-LIST-LOCATION

Meta-Entity-Substitute-Name = CODE-LOC
Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

COMMENTS

Maximum-Attribute-Length = 240
Minimum-Attribute-Length = 1

DATA-CLASS

Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

DATA-TYPE

Maximum-Attribute-Length = 16
Minimum-Attribute-Length = 5

DESCRIPTION

Meta-Entity-Substitute-Name = DESC
Maximum-Attribute-Length = 5000
Minimum-Attribute-Length = 1

DOCUMENT-CATEGORY

Meta-Entity-Substitute-Name = DOC-CAT
Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

DURATION-TYPE

Meta-Entity-Substitute-Name = DUR-TYPE

Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

DURATION-VALUE

Meta-Entity-Substitute-Name = DUR-VAL

Maximum-Attribute-Length = 22

Minimum-Attribute-Length = 1

EXTERNAL-SECURITY

Meta-Entity-Substitute-Name = SEC

Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

FREQUENCY

Meta-Entity-Substitute-Name = FREQ

Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

HIGH-OF-RANGE

Meta-Entity-Substitute-Name = HIGH

Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

INTERNAL-FORMAT

Meta-Entity-Substitute-Name = INTF

Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

JUSTIFICATION

Meta-Entity-Substitute-Name = JUS

Maximum-Attribute-Length = 5

Minimum-Attribute-Length = 4

LENGTH

Maximum-Attribute-Length = 22

Minimum-Attribute-Length = 1

LOCATION

Meta-Entity-Substitute-Name = LOC

Maximum-Attribute-Length = 32

Minimum-Attribute-Length = 1

LOW-OF-RANGE

Meta-Entity-Substitute-Name = LOW

Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

NUMBER-OF-LINES-OF-CODE

Meta-Entity-Substitute-Name = NO-LINES-CODE
Maximum-Attribute-Length = 22
Minimum-Attribute-Length = 1

NUMBER-OF-RECORDS

Meta-Entity-Substitute-Name = NO-OF-RECS
Maximum-Attribute-Length = 22
Minimum-Attribute-Length = 1

PRECISION

Maximum-Attribute-Length = 2
Minimum-Attribute-Length = 1

RECORD-CATEGORY

Meta-Entity-Substitute-Name = REC-CAT
Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

RELATIVE-POSITION

Meta-Entity-Substitute-Name = REL-POS
Maximum-Attribute-Length = 22
Minimum-Attribute-Length = 1

SCALE

Meta-Entity-Substitute-Name = SCL
Maximum-Attribute-Length = 2
Minimum-Attribute-Length = 1

SYSTEM-CATEGORY

Meta-Entity-Substitute-Name = SYS-CAT
Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

USAGE

Maximum-Attribute-Length = 32
Minimum-Attribute-Length = 1

IRDS-Defaults

EXISTING-IRDS-DEFAULTS

Format = STRING
Maximum-Attribute-Length = 32
Maximum-Entity-Assigned-Descriptive-Name-Length = 64


```
Maximum-Entity-Assigned-Descriptive-Name-Length-
                                         Default = 64
Maximum-Entity-Assigned-Access-Name-Length = 32
Maximum-Entity-Assigned-Access-Name-Length-Default = 32
Maximum-Number-Of-Occurrences = 10
Maximum-Number-Of-Occurrences-Default = 10
Minimum-Attribute-Length = 1
Minimum-Entity-Assigned-Descriptive-Name-Length = 1
Minimum-Entity-Assigned-Descriptive-Name-Length-
                                         Default = 1
Minimum-Entity-Assigned-Access-Name-Length = 1
Minimum-Entity-Assigned-Access-Name-Length-Default = 1
Significant-Attributes = 1
Standard-Mode = YES
```

IRDS-Limits

EXISTING-IRDS-LIMITS

```
Integer-Limit = 1000000000000000000000000  
Line-Count-Limit = 32767  
Line-Length-Limit = 80  
Maximum-Entity-Assigned-Access-Name-Length-Limit = 32  
Maximum-Entity-Assigned-Descriptive-Name-Length-Limit = 64  
Maximum-Meta-Entity-Assigned-Access-Name-Length-Limit = 32  
Maximum-Meta-Entity-Assigned-Descriptive-Name-Length-  
Limit = 64  
Maximum-Number-Of-Occurrences-Limit = 10  
String-Length-Limit = 72  
Variation-Name-Limit = 8
```

5. THE IRDS PROTOTYPE SOURCE CODE

5.1 OVERVIEW

The C language Prototype program translates IRDS commands into SQL commands and sends these to the Oracle database management system, where the database representing the IRD is maintained. The program performs various consistency checks, some of which include calls to the DBMS to access data. Formatting of the output and some of the entity selection is done at the C program level. The remainder of the selection is done through the DBMS facilities.

5.2 DICTIONARY SUBROUTINES

When the user executes the IRDS prototype, the C program looks for the Oracle table `DICTIONARY_NAMES` to get a list of available IRDs. If no such table is found, the subroutine `SET_DICT` will call `MK_DICT` to create the necessary tables. `MK_DICT` creates and fills `DICTIONARY_NAMES` and those tables that are fixed. `MK_DICT` also creates a set of tables that are modifiable, adding prefix `A_` to the name of each such table. `MK_DICT` then fills the new schema level tables, the data for which comes from the file `IRDS.TBL`. The IRD level tables are then created using the information contained in the schema level tables. The user is then asked to name the new IRD.

If the `DICTIONARY_NAMES` table exists but is empty, then the Prototype assumes that the static tables and a set of dictionary tables have already been created and filled. In this case, the user is asked to name the IRD.

If there is data in the `DICTIONARY_NAMES` table, then the list of IRDs is displayed to the user for the user's selection.

When the user executes a `CREATE IRD` command, the program executes the subroutine `CRE_DICT`, which finds a prefix to use and then creates a new IRD. The subroutine `SET_DICT`, which is responsible for making sure that the user is placed in the correct IRD, is executed before the user is given a prompt.

5.3 PARSING THE COMMANDS

Preliminary parsing of each IRDS command is performed by the subroutine GETCOM. GETCOM calls subroutines READCOM and INDEXCOMM. READCOM reads in a command from the standard input. INDEXCOMM takes the string of input from READCOM and divides it into words which are stored in the global array WORD. INDEXCOMM also determines which command was typed in, and records this in the global variable NCOMMAND.

Subroutine DO_COMMAND, called after GETCOM, calls subroutine CK_SYNTAX. CK_SYNTAX calls subroutine MATCH_TEMPLATE, giving it the template for the specific command and the array of words that INDEXCOMM produces. MATCH_TEMPLATE checks, word-by-word, that the template matches the array of words given. MATCH_TEMPLATE will not do any backtracking, instead counting on having unique choices when there are several options.

MATCH_TEMPLATE assumes that the following characters, when they appear in a template, mean special things:

[] { | } # ` '

These special meanings are as follows:

- o [and] surround a part of the command that is optional.
- o The construct { a | b | c } matches exactly one of a, b, or c, where a, b, and c do not have to be simple.
- o ` a ` will match 0 or more a's, where a does not have to be simple. The check for another a is made before the check for what comes after the ' in the template, and this should be considered when writing templates.
- o The character # is followed by a number, 1 through 9, which is the index to be used into an array of linked lists. The word at this position in the input is added to the linked list which has the given index.

Linked lists are used so that instances of the same type of structure can be stored together into fixed places in the array. For example, a list of attributes specified in an ADD or MODIFY command can all be in one place. These linked lists are dynamic, but because what is stored in them gets translated and stored into non-dynamic structures later,

there is a limit, about 100, to the number of items that can be in a list.

Output commands are not completely parsed by MATCH_TEMPLATE, which counts on subroutine WHERE_S to more thoroughly parse any WHERE clause. WHERE_S makes sure that the attribute-types used do exist, and also does other similar checks. WHERE_S uses backtracking to find the correct parse.

5.4 COMMAND SUBROUTINES

After a command has been read in and parsed, the linked list of values from the parse is passed by DO_COMMAND to the subroutine for that command. Each command has a corresponding subroutine, and each subroutine has, as its name, an abbreviation of the name of the command. The subroutines do the required consistency checking, and translate the command into a SQL command or a series of SQL commands, which are then executed. Examples of constraints that are checked are: modifying only existing entities, adding only one entity with a given access-name, and adding an attribute for an entity only if the entity's type is meta-related to the attribute's type with an entity-type-contains-attribute-type meta-relationship. Some of the checks involve retrieving information out of the Oracle database using SQL commands executed through subroutine calls. Some of the checks and actions are common to several commands, and thus have been written as separate subroutines.

5.5 OCI SUBROUTINES

The Oracle Call Interface, OCI, subroutines are the subroutines supplied by the DBMS. They all start with an O and are described in Oracle's Pro*C User's Guide. These subroutines allow SQL commands to be executed against a database in Oracle.

5.6 HLI SUBROUTINES

The Prototype's C program contains a special set of subroutines, the name of each member of which starts with HLI_. This is an attempt at a consistent interface to the DBMS that both eliminates the repeated writing of certain sequences of calls to Oracle's OCI subroutines, and also

checks for errors. Not all of the calls to the OCI subroutines in the rest of the code have been replaced, but the number has been reduced. This effort has helped to place the direct interface to the DBMS into a limited area of the source code.

5.7 GLOBAL VARIABLES

There are a few variables that were made global because of their frequent use in different subroutines. These global variables are defined at the top of each source code file. Two of the variables, CURSOR and LDA, were defined for the Oracle subroutines to use. WORD is an array of 100 strings that will hold the input after it has been split up into words. NWORDS is the number of words in the array WORD. PREFIX indicates which IRD a user has activated. NCOMMAND records the type of the current command (e.g., ADD ENTITY or OUTPUT IRD). There are a few global variables that are defined near the definition of a subroutine, and which are used only in that subroutine or set of subroutines.

5.8 PROGRAM DATA STRUCTURES

In each of the source code files, types are defined before the global variables are defined. Most of the types defined are structures. There are separate structures that store information about entities, relationships, and attributes, and similar ones that store information at the schema level.

There are a few static variables. The space for these is allocated in the global area, but the variables can be used only where they are defined. The static variables were used to save values between calls to a subroutine, without making the program responsible for the values.

Constants are defined in the file IRDS.CON. and are all in uppercase. One set of constants is used to allow the variable NCOMMAND to be assigned the name of a command instead of an integer or a string. Using an integer directly as the name of a command is confusing, and using a string would require a sequence of ELSE IF statements to determine which command subroutine to call. There is a set of constants to be used to set the length of strings, but these

constants have not been used consistently enough to allow them to be increased without the likelihood of problems arising.

6. INSTALLATION INSTRUCTIONS

The following are needed to install and run the Prototype:

1. A copy of the Oracle Database Management System
2. A "C" Compiler
3. Two 5 1/4 inch diskettes, supplied by ICST-NBS. These diskettes are written in DOS double-sided double-density format, and contain five ASCII text files. The files are:

```

irdsa.c  -)
irdsb.c  > --- the source code
irdsc.c  -)

irds.con  --- the settable constants

irds.tbl  --- the IRD-schema tables

```

To install the Prototype, the following steps should be performed in the order given:

1. Transfer the files from the diskettes to the host computer.
2. Choose or create an Oracle account for the IRDS tables.
3. Change the

```
#define ORACLE_UID "irds/irds"
```

statement in irds.con by replacing "irds/irds" with the Oracle userid/password to be used by the IRDS.

4. Change the

```
#define TABLEFILE "dra1:[kirk.irds.joe]irds.tbl"
```

statement in irds.con by replacing

```
"dra1:[kirk.irds.joe]irds.tbl"
```

with the complete name of the file that `irds.tbl` is stored in.

5. Compile `irdsa.c`, `irdsb.c`, and `irdsc.c`, using any standard "C" compiler. The Prototype uses Oracle version 4 or version 5 HLI subroutines, so the HLI libraries must be linked.
6. Run the executable. The first time it is run it will create and fill the tables it needs.

Other than in connection with 3 and 4 above, or in conjunction with a deliberate modification of the source code itself, it's probably not advisable to change any of the constants in `irds.con`. If you do change any of the constants, the source code must be recompiled. A newly compiled version can use the tables created by a previous version.

If you encounter any problems installing or using the Prototype, please contact Tammy Kirkendall at (301)975-3253 or Alan Goldfine at (301)975-3252.

REFERENCES

1. ANSI, American National Standard X3.138-1988, Information Resource Dictionary System, American National Standards Institute, New York, 1988.
2. Goldfine, A. H. and Konig, P. A., A Technical Overview of the Information Resource Dictionary System (Second Edition), NBSIR 88-3700, National Bureau of Standards, Gaithersburg, MD, January, 1988.
3. Goldfine, A. H., Using the Information Resource Dictionary System Command Language (Second Edition), NBSIR 88-3701, National Bureau of Standards, Gaithersburg, MD, January, 1988.
4. Law, M. H., Guide to Information Resource Dictionary System Applications: General Concepts and Strategic Systems Planning, NBS Special Publication 500-152, National Bureau of Standards, Gaithersburg, MD, April, 1988.
5. Newton, J. J., Guide on Data Entity Naming Conventions, NBS Special Publication 500-149, National Bureau of Standards, Gaithersburg, MD, October, 1987.

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 88-3830	2. Performing Organ. Report No.	3. Publication Date AUGUST 1988
4. TITLE AND SUBTITLE The ICST-NBS Information Resource Dictionary System Command Language Prototype			
5. AUTHOR(S) Alan Goldfine, Thomasin Kirkendall			
6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions) NATIONAL BUREAU OF STANDARDS U.S. DEPARTMENT OF COMMERCE GAITHERSBURG, MD 20899			7. Contract/Grant No. 8. Type of Report & Period Covered
9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)			
10. SUPPLEMENTARY NOTES <input type="checkbox"/> Document describes a computer program; SF-185, FIPS Software Summary, is attached.			
11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here) This publication is a report on the Information Resource Dictionary System (IRDS) Command Language prototype developed by the Institute for Computer Sciences and Technology of the National Bureau of Standards. It discusses the structure, source code, and operating environment of the Prototype, specifies the precise subset of the standard IRDS Command Language implemented, provides instructions for installing the Prototype software, and leads the reader through a typical user session.			
12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons) command language; data dictionary; data dictionary system; Information Resource Dictionary System; IRDS; prototype.			
13. AVAILABILITY <input checked="" type="checkbox"/> Unlimited <input type="checkbox"/> For Official Distribution. Do Not Release to NTIS <input type="checkbox"/> Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402. <input checked="" type="checkbox"/> Order From National Technical Information Service (NTIS), Springfield, VA. 22161			14. NO. OF PRINTED PAGES 55 15. Price \$13.95